ClickHouse кластер ctrl-c ctrl-v

Никита Михайлов





Обо мне

Никита, разработчик ClickHouse.



Мотивация

Кластер Яндекс.Метрики расположен на 500 серверах

В таблице visits хранятся сессии пользователей.

Это ~10PB данных в сжатом виде. ~100PB в несжатом.

Кластер не эффективен. Железо мощное и разное.

Нужно заменить железо на новое и снизить его количество.

Для этого нужно перенести все данные...



Как скопировать данные?

- Перенести данные сервера с одной машины на другую "руками"
- Используя встроенную функциональность
- Используя встроенную утилиту clickhouse-copier



Как ClickHouse хранит данные

```
$ tree ~/ClickHouse/db/ -d -C -L 1
    ├── config.d
    ├── data
    ├── dictionaries_lib
    ├── flags
    ├── metadata
    ├── preprocessed_configs
    ├── store
    └── users.d
```



Как ClickHouse хранит данные

Создадим базу с именем higload2021

```
CREATE DATABASE IF NOT EXISTS highload2021;
```



Зайдем в папку metadata и увидим там два новых файла: highload2021 и highload2021.sql

```
01. $ file highload2021*
    highload2021: symbolic link to
        ~/ClickHouse/db/store/bbd/bbdc8089-53c8-4d4a-9275-90f61bca08d5
    highload2021.sql: ASCII text


02. ↳ $ cat highload2021.sql
        ATTACH DATABASE _ UUID 'bbdc8089-53c8-4d4a-9275-90f61bca08d5'
        ENGINE = Atomic
```



Создадим в нашей базе таблицу

```
01. CREATE TABLE highload2021.participants(name String)
    ENGINE=MergeTree() ORDER BY tuple();

02. ↳ $ cat participants.sql
        ATTACH TABLE _ UUID '26d186a7-891f-4a87-a502-abda5a8fb028' (
            `name` String )
        ENGINE = MergeTree
        ORDER BY tuple()
        SETTINGS index_granularity = 8192
```

Теперь в папке higload2020 появился файл с метаданными таблицы



Вставим данные в таблицу:

```
INSERT INTO highload2021.participants
    VALUES ('Oleg Bunin'), ('Alexey Milovidov');
```

Посмотрим, куда сохранились данные нашей таблицы.

```
$ file  ~/ClickHouse/db/data/highload2021/participants
    ~/ClickHouse/db/data/highload2021/participants: symbolic link to
        ~/ClickHouse/db/store/26d/26d186a7-891f-4a87-a502-abda5a8fb028
```



Пройдем по symbolic link и увидим привычную директорию с данными для *MergeTree-таблиц

```
$ tree -L 2 ~/ClickHouse/db/store/26d/26d186a7-891f-4a87-a502-abda5a8fb028
├── all_1_1_0
│   ├── checksums.txt
│   ├── columns.txt
│   ├── count.txt
│   ├── data.bin
│   ├── data.mrk3
│   └── default_compression_codec.txt
├── detached
└── format_version.txt

2 directories, 7 files
```

Инструкция:

Ознакомиться со структорой папок на сервере А.

Скопировать все файлы на сервер В и "руками" сделать symbolic links.

Перезапустить сервер В. Поскольку инициализация таблиц происходит на старте.



Подводные камни:

1) Сервер может быть сконфигурирован с multi-volume storage, причем конфигурация сервера A может отличаться от конфигурации сервера B.

2) Версии серверов A и B могут сильно отличаться. В рассмотренном примере был движок баз данных Atomic. В версиях < 20.3 его еще не было.

3) Вы можете иметь дело с реплицируемыми таблицами, тогда помимо метаданных на диске нужно перенести сложную структуру метанных из Zookeeper.



Движок Distributed

Основная функциональность для создания шардов таблиц в ClickHouse.

Позволяет горизонтально масштабировать и storage, и compute.

Поскольку тяжелые вычисления (группировка, аггрегация) частично выполняются на шардах.



SELECT * FROM distributed_table

Distributed

SELECT * FROM local_table

SELECT * FROM local_table

Shard

Replica

Replica

Replica

Shard

Replica

Replica

Replica



Создадим Replicated-таблицу на каждом узле.
Макросы {shard}, {replica} указываются в конфиге каждого сервера.

```
CREATE TABLE hits_3 ON CLUSTER `{cluster}` (
    `Browser` String,
    `ClientID` UInt64,
    `EventDate` Date
)
ENGINE = ReplicatedMergeTree('/ch/tables/{shard}/hits_3', '{replica}')
PARTITION BY EventDate
ORDER BY (EventDate, intHash32(ClientID))
```

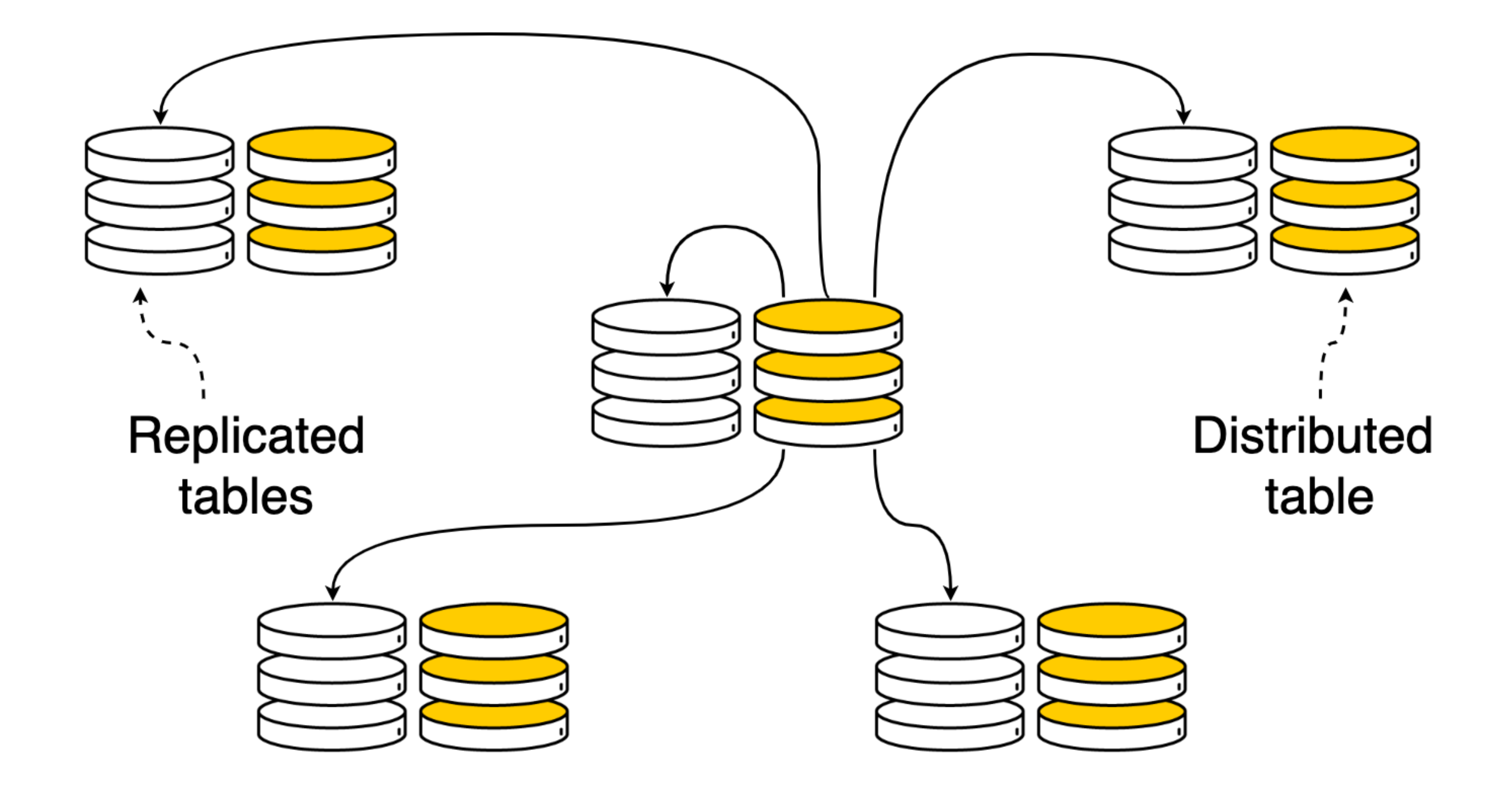


Создадим Distributed-таблицу на каждом узле кластера `{cluster}`.

```
CREATE TABLE hits_3_dist ON CLUSTER `{cluster}`AS db1.hits_3
ENGINE = Distributed('{cluster}', db1, hits_3, ClientID)
```

Таким образом можно обращаться к любому узлу и иметь доступ к данным всего кластера.




Replicated
tables
Distributed
table

Функциональность INSERT SELECT

```
01. CREATE TABLE destination ON CLUSTER remote_server AS source

02. INSERT INTO FUNCTION remote('127.0.0.1', currentDatabase(), destination)
    SELECT * FROM source
```



Функциональность INSERT SELECT

Табличная функция remote:
— создает неявную Distributed-таблицу;
— запрос выполняется над ней;

INSERT SELECT:
— создает пайплайн вычислений отдельно для INSERT и SELECT;
— склеивает их;

Итог:
— Данные передаются между серверами в Native-формате, то есть эффективно.



Копирование целого кластера

Source cluster

Copy?

Destination cluster



Копирование целого кластера

Source cluster

Distributed table

Destination cluster



Копирование целого кластера

Выберем произвольный узел и создадим две Distributed-таблицы:
— Первая смотрит на кластер source
— Вторая смотрит на кластер destination

```
INSERT INTO destination_distributed
    SELECT * FROM source_distributed;
```



Копирование целого кластера

Source cluster

Destination cluster

Data



Копирование целого кластера

Distributed-таблицы созданы на каждом узле обоих кластеров:
— Выбираем любой узел из source кластера
— Выбираем любой узел destination кластера

```
INSERT INTO FUNCTION
    remote('192.168.1.1', currentDatabase(), destination_distributed)
SELECT * FROM source_distributed;
```



Но есть проблемы...

— Что, если сеть моргнет при передаче данных?

— Копирование таким способом производится "в один поток".



Единицы измерения данных

— Гранула

— Блок (Чанк)

— Кусок

— Партиция

— Таблица

— Шард

— Кластер



Решение

Запустим отдельный INSERT SELECT для каждой партиции в исходной таблице.

Если получили ошибку – удаляем партицию в destination таблице и повторяем операцию.

Выглядит как алгоритм, поэтому этот процесс можно автоматизировать!



clickhouse-copier

Утилита, входящая в стандартную поставку clickhouse, позволяющая

— скопировать таблицу с одного кластера на другой;

— перешардировать кластер

clickhouse-copier умеет работать параллельно, используя Zookeeper для координации нескольких процессов



Конфигурация clickhouse-copier

Описывается в формате xml. Нужно указать:

— Узлы кластера Zookeeper

— Конфигурацию логгера

— Любые дополнительные настройки



Конфигурация задачи

Описывается в формате xml. Нужно указать:

— Конфигурацию source и destination кластера.

— Полное имя source и destination таблицы.

— Движок destination таблицы.

— Новый ключ шардирования.

— Условие для фильтрации данных при чтении.



Zookeeper
/copier/task
Source cluster
Destination cluster



HL
HighLoad++
Весна 2021

Запуск

```
clickhouse-copier
        --daemon
        --config /path/to/config.xml
        --task-path /task/path/zookeeper
        --task-file /path/to/task/on/disk
        --task-upload-force
        --base-dir /path/to/dir
```



Сложности

clickhouse-copier предполагает, что:

— Партиции не меняются в процессе.

— Схема таблиц сохраняется.

При партиционировании таблицы по месяцам копировать можно все партиции, кроме текущей.

Данные текущей партиции можно писать в оба кластера.



Рассматриваем алгоритм

Для начала нужно выяснить, какие партиции нужно копировать.

Для каждого шарда из source-кластера создадим Distributed-таблицу локально.
И выполним запрос.

```
SELECT DISTINCT $PARTITION_KEY_EXPRESSION$ AS partition
        FROM _local.`.read_shard_0.$TABLE_NAME$` ORDER BY partition DESC
```



Рассматриваем алгоритм

Пользователь может задать явно список подлежащих копированию партиций.

Это делается в секции enabled_partitions в конфигурации задачи.

```
<enabled_partitions>
    <partition>2021</partition>
</enabled_partitions>
```



Процесс копирования

Создадим destination-таблицу:

— Сделаем запрос SHOW CREATE TABLE на source-кластер;

— Преобразуем, изменив ENGINE, на новый, полученный из описания задачи;

— Выставим флажок, отвечающий за IF NOT EXISTS;

Последнее важно, поскольку пользователь мог сам создать таблицу, если колонки destination-таблицы отличаются.



Процесс копирования

Проитерируемся по всем партициям и скопируем их.

Сделаем запрос вида:

```
INSERT INTO destination_table_distributed
SELECT * FROM source_table_distributed
    WHERE PARTITION_KEY=CURRENT_PARTITION
```



Процесс копирования

Таблица destination_table_distributed смотрит на весь destination-кластер.

У этой таблицы можно задать ключ шардирования в описании задачи;

```
<sharding_key>
    rand()
</sharding_key>
```

Таким образом можно перешардировать кластер.



Проблема

Разбиение по отдельным партициям не всегда хорошо работает.

Живой пример – кластер Яндекс.Метрики, где одна партиция занимает сотни гигабайт.

Копирование такого объема данных неизбежно обернется ошибкой и все скопированные данные текущей партиции придется удалить из destination-таблицы.



График потребления диска на узле destination-кластера

y3
y2
y1
x1
x2
x3
x4
time



Решение

Отдельно копироваться должны не целые партиции, а их небольшие части (кусочки)!

Парты?

Нет.

— Множество партов может меняться из-за процесса фоновых слияний.

— Парт может быть достаточно большого размера.



Решение

Как поделить партицию примерно на равные части?

Посчитаем хэш от каждой строчки и возьмем остаток от деления на количество частей.

Для ускорения будем считать хэш не от всей строчки, а только от первичного ключа.

```
SELECT * FROM source_table
WHERE PARTITION_KEY = CURRENT_PARTITION AND
      cityHash64(PRIMARY_KEY) % N == 0
```



Альтернатива

У таблицы можно указать ключ семплирования.

Это специальная функциональность, чтобы делать запросы по некоторой части данных.

Можем воспользоваться ей, задавая запросы вида:

```
SELECT * FROM source_table

SAMPLE 1/N OFFSET CURRENT_PIECE_NUMBER/N

WHERE PARTITION_KEY = CURRENT_PARTITION
```



Конфигурация

Пользователь может задать количество частей в конфиге задачи.

```
<number_of_splits>
    42
</number_of_splits>
```



Сложности

Составные части партиции могут копироваться параллельно.

Что делать, если 9/10 частей партиции скопировались успешно, а при копировании последней вылетело исключение или любая другая ошибка?

Удалять всю партицию совсем не хочется...



Возможное решение

В случае неудачи можно запустить отдельный запрос вида:

```
ALTER TABLE destination_table DELETE
WHERE PARTITION_KEY = CURRENT_PARTITION AND
      cityHash64(PRIMARY_KEY) % N == 10
```

Но такой запрос будет выполняться долго, поскольку данные последнего кусочка партиции распределены неизвестным образом.



Текущее решение

Будем копировать кусочки во вспомогательные таблицы рядом с основной таблицей.

Например, если разбили исходную партицию на 10 частей, то получим 10 таблиц.

В i-ой таблице будут данные исходной таблицы, у которых хэш первичного ключа сравним с i по модулю 10.



Zookeeper

/copier/task

Source cluster

Destination cluster

0 1 2 3 4



Объединение данных

После успешного копирования партиции в auxiliary таблицы необходимо переместить данные в основную таблицу.

```
ALTER TABLE destination_table
    ATTACH PARTITION PARTITION_NAME  FROM auxiliary_table
```

Заметим, что в auxiliary-таблице партиция имеет такое же имя.



Синхронизация

Zookeer используется еще для многих вещей, например:

— Ограничение количества одновременно работающих процессов.

— Mutual exclusion для каждого кусочка партиции.

— Хранения результатов копирования для каждого кусочка (кто, когда)

— Хранения статуса копирования всех партиций таблицы.



Тестирование

Не было бы возможным без нашей замечательной инфраструктуры CI.

— Кластер ClickHouse поднимается в контейнерах.

— Процессы copier запускаются в случайных контейнерах.

По завершении всех процессов проверяются хитрые инварианты.



Fault injection

Очень дешевый способ для проверки отказоустойчивости программы.

В тестах задается вероятность того, что в случайный момент времени:

— Упадет копирование кусочка

— Упадет слияние всех кусочков воедино

Тестирование происходит 24/7...



Итог

clickhouse-copier - отказоустойчивый инструмент, который позволяет копировать большие объемы данных ClickHouse.

Настроил и забыл.

С помощью него был скопирован не один кластер Яндекс.Метрики.



Спасибо!

Контакты:

Telegram: @nikitamikhaylov

email: jakalletti@yandex-team.ru

